CHEETAHBASE.COM

CarsonDB™

Users Guide

Version 1.0.1

Update May 12, 2018

Table of Contents

Summary

CarsonDB is a high-performance library for querying data in the AVImark veterinarian practice management software. AVImark is the industry leader in veterinarian practice management software. CarsonDB is simply the fastest solution for solutions needing access to AVImark data. CarsonDB contains Open Source and freeware components. The main interface library is written using the Microsoft .NET Framework. The data mining library is written using x86 Assembly language. All tools are currently compatible with the Microsoft Windows family of operating systems.

CarsonDB is fast because it uses Assembly language for the fastest performance possible. CarsonDB does not use indexes because they may be unreliable or their functionality may change over time. Since some AVImark database tables may be over a gigabyte in size, especially in larger practices, performance is of upmost importance. This necessitates the use of the lowest-level language possible to make sure all integrated systems have the best performance possible.

Whom This Guide Is For

This user's guide is for a software developer or programmer who is writing software that needs access to the AVImark databases. The developer needs to be reasonably familiar with C# and Microsoft .NET programming. It is not in the scope of this guide to explain the basics of software development.

The developer should be reasonably familiar with the operation of the AVImark software including specific AVImark terminology. It is beyond the scope of this guide to explain how the AVImark system works as a whole. This document describes how use the CarsonDB data objects which allows access directly to the AVImark databases.

The developer should have access to working copies of AVImark databases. No sample databases are included with this product. If data is not immediately available, it may be best to form relationships with veterinary practices that use the AVImark software who would be willing to give you access to their data. Another option is to purchase a licensed copy of AVImark software.

Features

Fast AVImark Database Access

CarsonDB utilizes Assembly language for the fastest performance possible. The query engine, database layer and difference engine all use Assembly language.

Fast Data Difference Engine

CarsonDB has a built-in data difference engine. This gives integrated systems unparalleled access to record changes. Even on files over a gigabyte, it takes seconds to find out what records have changed since the last data pull. CarsonDB does this by calculating a CRC checksum on the desired fields. CarsonDB will find these changes quickly and report on them. No third party controls needed because CarsonDB has everything needed to do this natively.

Open Source and Freeware Components

CarsonDB uses code written in Microsoft C# and integrates well with systems that utilize the Microsoft .NET framework. The C# code is Open Source and is licensed under the MIT License. The Assembly language code is freeware, but it is currently closed source. It will remain closed source because GST Computing, LLC does not wish to expose AVImark's proprietary database format. Please see the licensing sections for more information.

Thorough and Automated Testing

The CarsonDB source code contains automated unit tests for all tables in addition to generic filtering and multi-threaded data access to ensure that each version of the library is as error free as possible.

Technical

CarsonDB is compiled as a 32-bit application purposely. The data access engine, CarsonLib.DLL, is written in x86 32-bit Assembly language. Currently, a 64-bit version of this DLL has not been developed. This should not be an issue because all Windows systems will support both 32-bit and 64-bit software. Since the code is memory-efficient, huge amounts of RAM is not necessary. A 64-bit version may be developed in the future.

CarsonDB only supports a subset of the entire AVImark database. Many of the database objects are not necessary, but more tables and fields will be available in the future. These tables and fields will be based on requests and availability of resources. To request new tables or fields, visit the CarsonDB support forums located at CheetahBase.com.

CarsonDB is written to be thread safe by utilizing locks in code. There are also two automated Unit Tests that specifically test for thread safety. However, it is up to the individual developer to ensure thread safety through proper testing.

Data Filtering

The fast speed of CarsonDB comes from data filtering. Data filtering should happen before using LINQ queries. This brings back fewer records from the database. If a simply LINQ query is used, all records must be returned so that the LINQ engine can determine if a record should be used or not. Consider the following code fragments. The first example uses LINQ only while the second one uses filters.

Example without filtering:

```
Account account = new Account(@"C:\AVImark");
var accounts = account.AccountList();

var result = from a in accounts
             where a.AccountDate > new DateTime(2015, 1, 1)
             select a;

int recordCount = result.Count();
```

Example with filtering:

```
Account account = new Account(@"C:\AVImark");
account.AddFilterCriteria(Account.AccountFields.AccountDate, ComparisonType.GreaterThan,
       new DateTime(2015, 1, 1));
var accounts = account.AccountList();

int recordCount = accounts.Count();
```

While the differences in code seem small, one test showed (with a gigabyte-plus Account file) a significant performance improvement in the second example. The results were as follows:

First example total elapsed time: 25.1600128 seconds

Second example total elapsed time: 9.7162344 seconds

The first example, while not an especially bad speed considering the amount of data, is improved almost 300% by adding a filter instead of doing the query work through LINQ.

Important

All filters are added together. They all use the logical "AND" operator. That means if two or more filters are applied, only the results that match both filters are returned.

If the field is a string type, the filtering options are different. The valid filters for a string are ComparisonType.EqualTo, ComparisonType.NotEqualTo and ComparisonType.Contains. ComparisonType.Contains is unique to string types which is not valid for other data types.

Any field that is a Boolean or Bit cannot be used as a filter. Any attempt to add a filter to an invalid field will throw an exception.

Deleted Records

AVImark keeps a record status on all fields. The fields name ends with "RECD". For instance, on the Accounts table, it is AccountRecd. If the status is "D", the record is deleted. The record should not be used or considered valid. CarsonDB filters these records out automatically. However, if there is a reason that this data should be visible, there is a public property that will turn this off.

```
CarsonStaticSettings.FilterActiveRecords = false;
```

Also, if a filter already exists that uses the RECD field, the additional RECD checking will be skipped.

Locked Records

AVImark keeps records locked during normal operation. This is normal for any database system to do to prevent concurrency issues. However, this may pose a problem when attempting to read these records. All tables have a field named RecordState. This is based on an enum named RecordStatus. The value will always be "Error" when a record is locked or is otherwise unreadable.

Getting Started

CarsonDB uses the "CarsonDB" namespace. Any program that accesses the library needs to either include a "using CarsonDB;" statement at the beginning of each file that uses it, or all calls should be proceeded with CarsonDB. For example:

```
CarsonDB.Animal animal = new CarsonDB.Animal(@"C:\AVImark");
```

Bear in mind that AVImark database files are files that exist individually in the file system. It is important for an application to know where the files reside. It is typical for the files to exist in the c:\AVImark folder on the server it is installed on. The CarsonDB library must know where the files exist to work. There are two ways to initialize this path. The first way is to instantiate the first object with the path name. For example:

```
Animal animal = new Animal(@"C:\AVImark");
```

Please note that all table classes implement `IDisposable` so it is best to wrap them in `using` blocks.

When using the method above, only the first instantiation needs the path. On all subsequent calls, it is acceptable for the initialization to be:

```
Service service = new Service();
```

The other method is to set the following property:

```
CarsonStaticSettings.DatabasePath = @"C:\AVImark";
```

After that, all calls can be made without initializing with a path.

```
Animal animal = new Animal();

Service service = new Service();

etc.
```

Retrieving Data

To get data from a table, it is a simple and straightforward process.

- Instantiate the table object
- Set database filters
- Execute the "List" function for the table

From there, the fields can be looped through to retrieve all values. Below are some examples to help get started.

```
Service service = new Service(@"C:\AVImark");
service.AddFilterCriteria(Service.ServiceFields.ServiceDate, ComparisonType.GreaterThan,
new DateTime(2004, 1, 1));

var services = service.ServiceList();

foreach (var s in services)
{
	Console.WriteLine(s.Id);
	Console.WriteLine(s.ServiceAmount.ToString());
	Console.WriteLine(s.ServiceNote);
}
```

Complex queries using LINQ work similarly. The following example joins the Animals and Clients tables together to get the combinations, with filters on both tables.

```
Client client = new Client(@"C:\AVImark");
Animal animal = new Animal();
```

```
client.AddFilterCriteria(Client.ClientFields.ClientLast, ComparisonType.EqualTo,
       "Smith");
animal.AddFilterCriteria(Animal.AnimalFields.AnimalAdded,
       ComparisonType.GreaterThanEqual, new DateTime(2015, 1, 1));

var clients = client.ClientList();
var animals = animal.AnimalList();

var members = from a in animals
              join c in clients on a.AnimalClient equals c.Id
              select new { ClientName = c.ClientFirst + " " + c.ClientLast, AnimalName =
              a.AnimalName };

foreach (var m in members)
{
       Console.WriteLine("Client: " + m.ClientName + " Animal: " + m.AnimalName);
}
```

Difference Engine

CarsonDB has an advanced difference engine written using Assembly language. This guarantees that new and updated records can be found within seconds. CarsonDB does not depend on third party libraries to locate new records. All of the functionality to locate records is free of other external dependencies. The difference engine saves a hash of all records for the requested fields and saves it to a file. On the next run, it compares the hashes.

Why would a difference engine be needed? Some applications, such as an application that stores veterinarian data in the cloud, need to synchronize data. It would be a waste of time and resources to upload all records every time. The difference engine locates all records that have changed since the last call.

CarsonDB makes this easy. There is a function called RecordsChangedSinceLastSnapshot. This function makes it relatively easy to locate added and updated records with minimal effort. The first time the function is called, a new difference file is created, and all records are returned. All subsequent calls will return added and updated. One word of caution, if the difference file is deleted all records will be returned. The calling application should be able to handle that contingency. Here is an example:

```
Account account = new Account(@"c:\AVImark");
var accounts = account.RecordsChangedSinceLastSnapshot<Account.AccountData>();
```

Every table class has the ability to perform this function. The type passed to the table will be the class name plus the word "Data" trailing it. For instance, the following would apply to the Animal table.

```
Animal animal = new Animal(@"c:\AVImark");
var animals = animal.RecordsChangedSinceLastSnapshot<Animal.AnimalData>();
```

Also, bear in mind that this function will return "D" values for the RECD fields. This is necessary because some records may change from active to deleted and the calling application would have to react accordingly.

There are several overloads for this function. If you pass in no parameters, all fields are used to determine if a record has changed. If preferable to trigger a change based on select fields, here is an example of how to perform this:

```
accounts = account.RecordsChangedSinceLastSnapshot<Account.AccountData>(new List<string>
        { Account.AccountFields.AccountAnimal, Account.AccountFields.AccountClient,
        Account.AccountFields.AccountCode });
```

In this case, only a change in AccountAnimal, AccountClient and AccountCode fields would trigger a change. However, this does not affect new records. New records are always returned.

If it is important to know if the record was updated or added, all tables have a field named RecordState. This is based on an enum named RecordStatus. The state will be Added for added record and Updated for updated records. It may be Error, which indicates an error (most likely a record locked error) occurred when trying to read the record.

Caution

Please be aware that the difference engine depends on the previous difference file being present in either the specified path or the default location of the application if a path is not specified. The difference file will be the table name with the extension of "crc". For instance, the Accounts table would be "Accounts.crc". If the file ever gets deleted, the next comparison would bring back all records. Most of the time, this would be an undesirable situation unless the goal is to completely resynchronize data. The application needs to be aware of this condition and react according.

Licensing Information

CarsonLib is the Assembly language DLL referred to in various portions of this document. CarsonLib is freeware, but it is not Open Source. The license is detailed below.

CarsonDB is the Open Source library created in Microsoft C#. CarsonDB is licensed with the MIT Open Source license.

Information on the licenses and restrictions are listed below.

CarsonDB License

CarsonLib License

CarsonLib Veterinary Extraction Library

CarsonLib.DLL is free for use in any environment, including but not necessarily limited to: personal, academic, commercial, government, business, non-profit, and for-profit. "Free" in the preceding sentence means that there is no cost or charge associated with the use of CarsonLib.DLL.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software (the "Software"), to use the Software without restriction, including the rights to use, copy, publish, and distribute the Software, and to permit persons to whom the Software is furnished to do so.

You may not modify, adapt, rent, lease, loan, sell, or create derivative works based upon the Software or any part thereof. For the purposes of this license, a derivate work is a software package, library, or program with similar functionality to this library or would be in direct competition with the functionality of this library or API therein.

This notice must be included in the same directory as the CarsonLib.DLL library file.

The Copyright notice below should be included in the “About Dialog” of any software program and/or any documentation that accompanies any program that utilizes CarsonLib.DLL.

> CarsonLib Veterinary Extraction Library
>
> Copyright ©2018 GST Computing, LLC. All Rights Reserved.
>
> For more information, visit CheetahBase.com.

The laws of the state of Oklahoma, USA, govern this software license.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.